第三章 补充案例

案例3-1 函数定义与加载

一. 案例描述

1. 考核知识点

   函数的调用

2. 练习目标

   - 掌握函数的定义与调用之间的关系

3. 需求分析

在 PHP 中，程序对函数是进行预加载的，因此在同一个文件中，定义和调用函数没有先后之分。但是在调用包含文件中的函数时，定义和调用函数有无先后之分呢？

接下来在 function.php 中定义一个获得最小值的函数 getMin($m,$n)，然后通过测试文件 index.php 先调用 getMin()函数，再包含 function.php 文件，查看运行结果。

4. 设计思路（实现原理）

   1） 在 function.php 中定义一个获得最小值的函数 getMin($m,$n)。

   2） 在测试文件 index.php 中，先调用 getMin()函数，再包含文件 function.php。

二. 案例实现

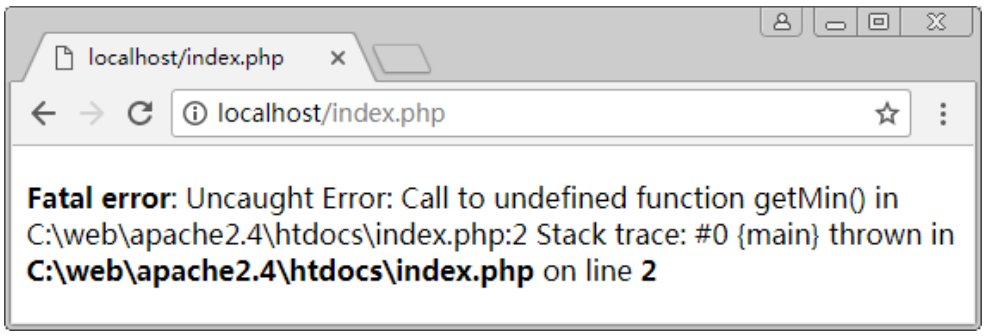
1. 在 function.php 文件中，定义 getMin($m,$n)，代码如下：

```
<?php
function getMin($m, $n)
{
    return $m < $n ? $m : $n;
}
```

2. 测试文件 index.php，代码如下：

```
<?php
echo getMin(23, 17);
include './function.php';
```

运行程序，结果如下图所示。



3. 当函数定义与调用不在同一个文件中时，若调用函数在包含前未定义，会出现函数未定义错误。

三. 案例总结

1. 在同一文件中，可以先调用后定义函数。
2. 如果调用的函数在包含文件中，不能先调用。这是因为 include 语句只有在被执行时才会读入要包含的文件，即定义函数文件。

案例3-2 函数的返回值

一. 案例描述

1. **考核知识点**
   函数的返回值
2. **练习目标**
   - 掌握函数返回值的各种情况
3. **需求分析**
   在 PHP 中，使用 return 语句可以将返回值传递给调用者，并且返回值可是变量、常量或表达式等。为了更好的理解函数的返回值，下面定义一个没有 return 语句的函数，或者有 return 语句但没有写返回值，以及多个 return 语句的情况，观察函数的返回结果。
4. **设计思路（实现原理）**
   1）编写一个函数 getAge()，让其没有 return 语句。
   2）编写一个函数 getName()，让其 return 后面没有值。
   3）编写一个函数 getGrade($score)，判断传递过来的$score，当其大于 60 时，返回 pass，否则返回 fail。

二. 案例实现

1. 创建 PHP 文件：index.php，完成函数的定义。代码如下：

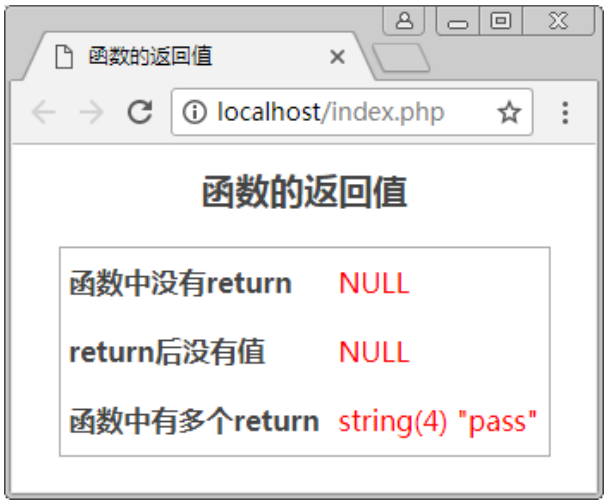
```
<?php
    // 函数中没有 return
    function getAge() {

    }
```

```
        // return 后没有值
        function getName(){
            return;
        }
        // 函数有多个返回值
        function getGrade($score){
            if ($score > 60) {
                return 'pass';
            }
            return 'fail';
        }
    ?>
```

2. 继续编辑 PHP 文件，index.php，展示调用函数后的返回值。代码如下：

```
    <!doctype html>
    <html>
      <head>
        <meta charset="utf-8">
        <title>函数的返回值</title>
        <style>
          h1{font-size:20px;color:#444;text-align:center}
          .tbl{border-collapse:collapse;color:#ff0000;border:1px solid
#aaa;margin:20px auto}
          .tbl th{font-weight:bold;color:#444;text-align:left;padding:5px}
          .tbl td{height:30px;line-height:30px;padding:5px}
        </style>
      </head>
      <body>
        <h1>函数的返回值</h1>
        <table class="tbl">
          <tr>
            <th>函数中没有 return</th>
            <td><?php var_dump(getAge());?></td>
          </tr>
          <tr>
            <th>return 后没有值</th>
            <td><?php var_dump(getName());?></td>
          </tr>
          <tr>
            <th>函数中有多个 return</th>
            <td><?php var_dump(getGrade(75));?></td>
          </tr>
        </table>
      </body>
    </html>
```

在浏览器中运行该 PHP 文件，结果如下图所示：



三. 案例总结

1. 函数没有 return 时，通过 var_dump()打印后得到的值为 NULL。
2. 函数有 return，但是后面没有数据，也会得到 NULL。
3. 函数代码中可以有多个 return。
4. 函数体代码运行到 return，那么后面的代码就不会再被执行，函数将结束运行。

案例3-3 参数数量

一. 案例描述

1. **考核知识点**

   参数数量

2. **练习目标**

   ➢ 了解调用函数时参数的数量问题

3. **需求分析**

   定义含有 3 个形参的函数 sayHello()，并为第 3 个形参设置默认值，对比如下 4 种情况的结果。

   - 调用此函数传递相同数量的实参。
   - 调用此函数传递多于形参的实参。
   - 调用此函数时省略有默认值的参数。
   - 调用此函数传递少于形参的实参。

4. **设计思路（实现原理）**

   1） 定义一个函数 test($a, $b, $c='TEST')。

   2） 使用不同数量的实参调用 test()函数。

   3） 对比并观察输出结果。

二. 案例实现

1. 编写 PHP 文件：index.php，用于定义函数 test()。代码如下：

```
<?php
    // 定义函数
    function test($a, $b, $c='TEST')
    {
        echo $a, $b, $c, '<br>';
    }
?>
```

2. 继续编辑 PHP 文件，展示传递不同参数的调用结果。代码如下：

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>参数数量</title>
    <style>
      h1{font-size:20px;color:#444;text-align:center}
      .tbl{border-collapse:collapse;height:30px;line-height:30px;color:#f00;m
argin:20px auto;width:700px}
      .tbl th{border:1px solid #aaa;font-weight:bold;color:#444;width:220px}
      .tbl td{border:1px solid #aaa;padding:0 10px}
    </style>
  </head>
  <body>
    <h1>参数数量</h1>
    <table class="tbl">
      <tr>
        <th>实参情况</th>
        <td>运行结果</td>
      </tr>
      <tr>
        <th>传递相同数量的实参</th>
        <td><?php test('1234', 'php', '---'); ?></td>
      </tr>
      <tr>
        <th>传递多于形参的实参</th>
        <td><?php test('1234', 'php', '---', 'very good.'); ?></td>
      </tr>
      <tr>
        <th>省略可选参数</th>
        <td><?php test('1234', 'php'); ?></td>
      </tr>
```

```
        <tr>
          <th>省略必选参数</th>
          <td><?php test('itcast'); ?></td>
        </tr>
      </table>
    </body>
</html>
```

运行程序，结果如下图所示：

三. 案例总结

1. 正常情况函数调用使用的实参数量是和函数定义的形参数量是一致的。
2. 如果调用的实参比形参数量少，未传递的实参对应的形参（且是靠右的）有默认值时，可以省略，不是这种情况，则会出错。
3. 调用函数的实参的数量比形参多的话，不会有影响。

案例3-4 函数中变量的作用域

一. 案例描述

1. **考核知识点**
   函数中变量的作用域
2. **练习目标**
   - 掌握全局变量在函数内使用的方法
3. **需求分析**
   全局变量是不能直接在函数内部使用的，需要使用 global 关键字，或使用$GLOBALS 超全局变量。

假设在函数外定义一个$radius 变量，然后在 getArea()函数中直接使用$radius，与在函数 getArea()中使用关键字 global 声明$radius，或是使用$GLOBALS 进行对比。

4. **设计思路（实现原理）**

1）编写 PHP 文件，获取圆的半径$radius。
2）编写 getArea()函数，直接使用半径$radius，计算圆的面积。
3）在函数中使用 global 关键字声明半径$radius，计算圆的面积。
4）使用$GLOBALS 获取半径$radius，计算圆的面积。
5）对比输出结果。

二. 案例实现

1. 编写 PHP 文件：index.php，完成 HTML 页面的设计。代码如下：

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>函数中变量的作用域</title>
    <style>
      h1{font-size:18px;text-align:center}
      .box{width:350px;border:2px dotted #FE0D0B;padding:0 10px 10px
10px;margin:20px auto}
    </style>
  </head>
  <body>
    <div class="box">
    <h1>函数中变量的作用域</h1>
    <?php
        // 此处编写 PHP 代码
    ?>
    </div>
  </body>
</html>
```

2. 继续编辑 PHP 文件，用于直接使用半径$radius，计算圆的面积。

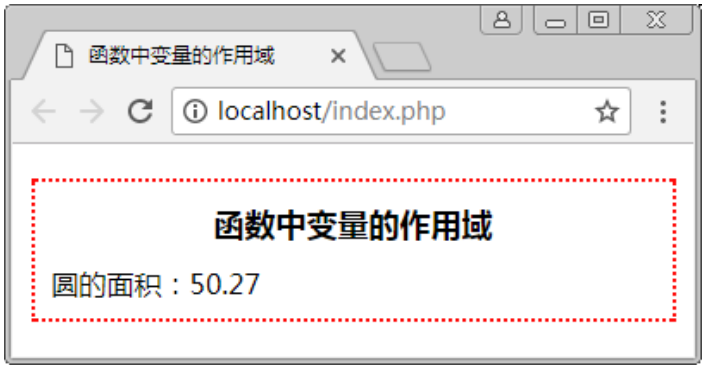
```
<?php
    function getArea()
    {
        $area = round(pi() * pow($radius, 2), 2);
        return $area;
    }
    $radius = 4;    // 圆的半径
    echo '圆的面积：' . getArea();
}
```

运行程序，结果如下图所示：

3. 修改 getArea()函数，在函数中使用 global 关键字声明$radius。代码如下：

```
function getArea()
{
    global $radius;
    $area = round(pi() * pow($radius, 2), 2);
    return $area;
}
```

运行程序，结果如下图所示：



4. 修改 getArea()函数，使用$GLOBALS 获取$radius。代码如下：

```
function getArea()
{
    $radius = $GLOBALS['radius'];
    $area = round(pi() * pow($radius, 2), 2);
    return $area;
}
```

运行程序，运行结果与上一步相同。

三. 案例总结

在函数内部使用外部变量时，可以通过如下 3 种方式实现：

1. 通过参数传递，传值或传引用。
2. 通过 global 关键字来提升。

3. 通过$GLOBALS 来获取。

案例3-5 静态变量与引用传参

一. 案例描述

1. **考核知识点**
   静态变量
2. **练习目标**
   - 掌握静态变量在函数中的使用
3. **需求分析**

在 PHP 中，函数内声明的静态变量

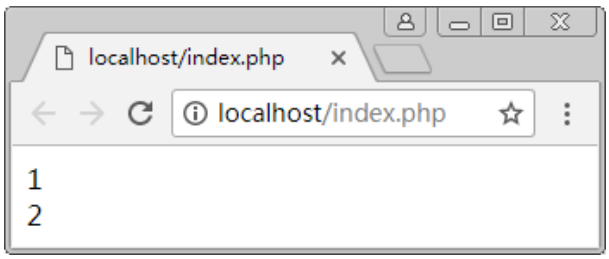
4. **设计思路（实现原理）**
   1）在函数中使用 static 定义静态变量
   2）通过“&”符号在函数外部引用函数中的静态变量

二. 案例实现

1. 编写 index.php，具体代码如下。

```
<?php
function &test()
{
    static $a = 1;
    echo $a, '<br>';
    return $a;
}
$num = &test();
$num = 2;
test();
```

2. 运行程序，结果如下所示。



三. 案例总结

1. 静态变量中的值在函数执行完成后不会释放。
2. 函数内的静态变量可以在函数外通过引用方式修改。

案例3-6 递归函数

一. 案例描述

1. 考核知识点

   递归函数

2. 练习目标

   ➢ 熟练掌握递归函数的妙处

3. 需求分析

递归函数通常有很高的使用价值，常用来将复杂的问题分解为简单的并相同的情况，反复做这种处理直到问题解决。假设有一只猴子摘了一堆桃子，当即吃了一半，可是桃子太好吃了，它又多吃了一个，第二天它把第一天剩下的桃子吃了一半，又多吃了一个，就这样到第十天早上它只剩下一个桃子了，问它一共摘了多少个桃子？

4. 设计思路（实现原理）

   1） 编写一个函数 peach($n)，每次都判断一下$n 是否等于 1。

   2） 如等于 1，则返回 1。

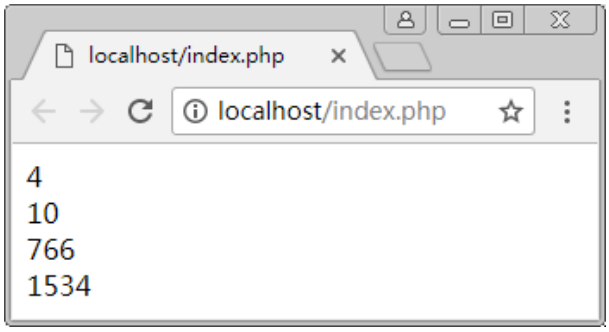
   3） 如不相等，则 $n - 1，再调用自身 peach()，并传递$n - 1 的结果。

二. 案例实现

1. 编写 index.php，代码如下

```
<?php
function peach($n)
{
    if ($n == 1) {
        return 1;
    }
    return 2 * (peach($n - 1) + 1);
}
echo peach(2), '<br>';
echo peach(3), '<br>';
echo peach(9), '<br>';
echo peach(10), '<br>';
```

2、运行程序，结果如下图所示：



三. 案例总结

1. 递归函数必须要有结束的条件，否则就是死循环。
2. 一定要找到合适的递归条件。
3. 递归函数的运算量很大，所以要谨慎使用。

案例3-7 strcmp()函数

一. 案例描述

1. 考核知识点

strcmp()函数

2. 练习目标

- 掌握 strcmp()函数的用法

3. 需求分析

PHP 提供的内置的字符串函数，功能非常强大，很容易就能实现各种业务逻辑，减少编程的难度。其中 strcmp()函数可以对字符串进行比较操作（以二进制形式）。

下面实现比较两次输入的密码（由字母和数字组成）是否正确。

4. 设计思路（实现原理）

1） 通过变量保存两次输入的密码值（由字母和数字组成）。

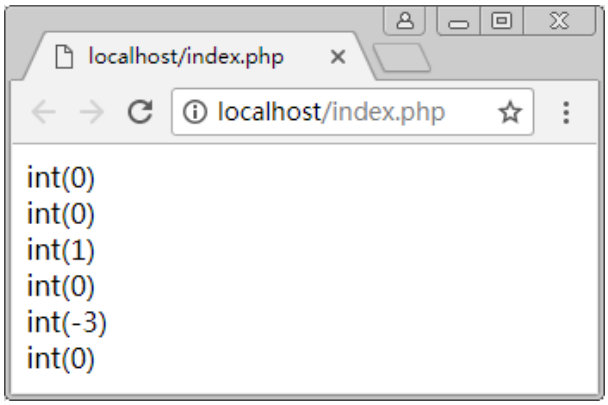
2） 判断两次输入的密码是否相等，并输出相应的提示语句。

二. 案例实现

1. 编写 index.php，代码如下：

```
<?php
var_dump(strcmp('test', 'test'));
echo '<br>';
var_dump(strcmp('0', 0));
echo '<br>';
var_dump(strcmp('test', 0));
echo '<br>';
var_dump(strcmp('', NULL));
echo '<br>';
var_dump(strcmp(NULL, '123'));
echo '<br>';
var_dump(strcmp('', false));
```

2. 运行程序，结果如下图所示：



三. 案例总结

在 PHP 中，可以通过调用 strcmp()函数进行字符串的比较。

案例3-8 substr()函数

一. 案例描述

1. 考核知识点

substr()函数

2. 练习目标

- 掌握 substr()函数的用法

3. 需求分析

在程序开发中，经常需要截取一个字符串中的某一部分，也就是获取字符串中的某个子串。PHP 提供了 substr()函数用来获取字符串的子串。

下面实现获得某个地址字符串中 image 首次出现的位置到字符串结尾的子字符串。

4. 设计思路（实现原理）

1）准备一个路径（其中要含有 image）。

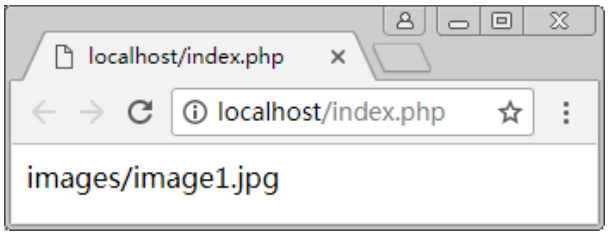
2）获取该路径中某个字符串首次出现的位置到结尾的子字符串，并将其返回。

二. 案例实现

1. 编写程序，代码如下：

```
<?php
$image = 'C:/web/example.com/images/image1.jpg';
echo substr($image, strpos($image, 'image'));
```

2. 运行程序，结果如下图所示：



三. 案例总结

在 PHP 中，substr()函数经常与 strpos()函数配合使用。

案例3-9 str_repeat()函数

一. 案例描述

1. 考核知识点
   str_repeat()函数
2. 练习目标
   - 掌握 str_repeat()函数的用法
3. 需求分析
   实现手机号的部分隐藏，将其从第 3 位到后 4 位之间的数字使用“*”代替。
4. 设计思路（实现原理）
   1） 随意编写一个手机号。
   2） 使用 str_repeat()重复长度为 4 的“*”字符串。
   3） 使用 substr_replace()函数将手机号中除前 3 位和后 4 位的数字使用“*”代替。

二. 案例实现

1. 编写 PHP 文件：index.php，用于使用“*”代替电话号中除去前 3 位和后 4 位的数字。代码如下：

```
<?php
    // 手机号
    $tel = '15811012345';
    // 获取 **** 字符串
    $replace = str_repeat('*', 4);
    // 实现手机号部分隐藏
    $new = substr_replace($tel, $replace, 3, 4);
?>
```

2. 继续编写 PHP 文件，在 HTML 页面中展示处理结果。代码如下：

```
<!doctype html>
<html>
  <head>
```

```
        <meta charset="UTF-8">
        <title>隐藏手机号</title>
        <style>
          .box{width:240px;padding:20px;border:2px dotted
#222;background:#f7fcfc;margin:0px auto;line-height:40px;color:#0066b3}
        </style>
      </head>
      <body>
         <div class="box">
           ☏ 原电话号：<?php echo $tel; ?> <br>
           ☏ 现电话号：<?php echo $new; ?>
        </div>
      </body>
    </html>
```

运行程序，结果如下图所示：

三. 案例总结

在 PHP 中，可以通过调用 str_repeat()函数，可以达到重复某个字符串的效果。

案例3-10　trim()、ltrim()和 rtrim()函数

一. 案例描述

1. 考核知识点

   trim()函数

2. 练习目标

   ➢ 掌握 trim()、ltrim()和 rtrim()函数的用法

3. 需求分析

使用 trim()函数对字符串中的制表符进行过滤操作。

4. 设计思路（实现原理）

   1） 编写前后一个带有制表符的字符串。

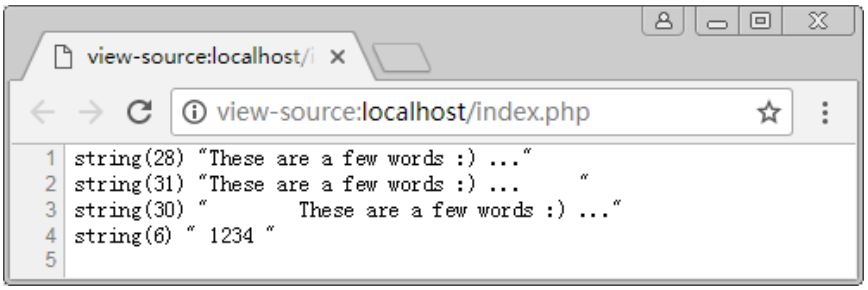
2） 使用 trim()对字符串进行过滤，并将结果返回。

二. 案例实现

1. 编写 index.php，代码如下：

```
<?php
$text = "\t\tThese are a few words :) ... \t ";
var_dump(trim($text));
var_dump(ltrim($text));
var_dump(rtrim($text));
$text = 'eessees 1234 ss';
var_dump(trim($text, 'es'));
```

2. 运行程序，查看源代码，结果如下图所示：



三. 案例总结

通过 trim()函数可以将字符串中的空白字符（空格符、制表符、换行符）等特殊字符过滤掉。ltrim()只对字符串左边过滤，rtrim()只对字符串右边过滤。过滤字符也可以指定为其他字符。

案例3-11 BCMath 数学运算

一. 案例描述

1. **考核知识点**

数学函数

2. **练习目标**

- 掌握利用 BCMath 相关函数进行数学运算

3. **需求分析**

在 PHP 中，由于整型表示的范围有限，以及浮点数运算存在精度问题，PHP 提供了 BCMath 数学扩展，用于实现利用字符串表示任意大小和精度的数字的二进制计算。

4. **设计思路（实现原理）**

1） 通过 bccomp()函数比较两个数的大小。

2） 通过 bcadd()函数实现加法运算。

3） 通过 bcsub()函数实现减法运算。

4） 通过 bcmul()函数实现乘法运算。

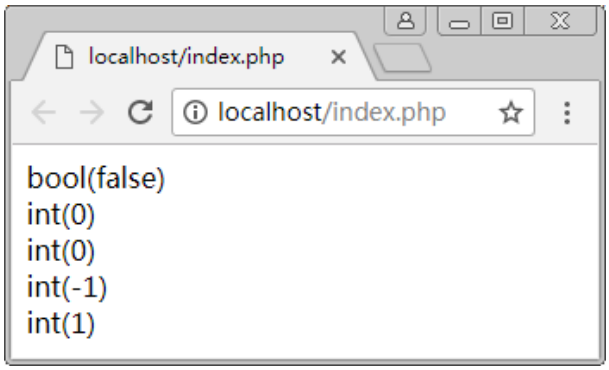
5） 通过 bcdiv()函数实现除法运算。

二. 案例实现

1. 编写 index.php，具体代码如下。

```
<?php
var_dump(0.1 == (1 - 0.9));
echo '<br>';
var_dump(bccomp(0.1, 1 - 0.9));          // 相等返回 0
echo '<br>';
var_dump(bccomp(0.123, 0.125, 2));       // 只比较 2 位小数
echo '<br>';
var_dump(bccomp(0.123, 0.125, 3));       // 小于时返回 -1
echo '<br>';
var_dump(bccomp(0.125, 0.123, 3));       // 大于时返回 1
```

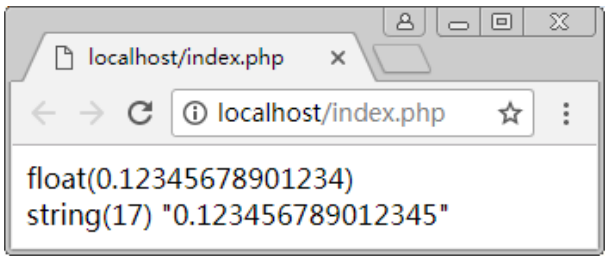
运行程序，结果如下图所示。



2. 利用 bcadd()进行加法运算，参数可以用字符串来表示。

```
var_dump(0.123456789012345);
echo '<br>';
var_dump(bcadd('0.123456789012345', '0', 15));
```

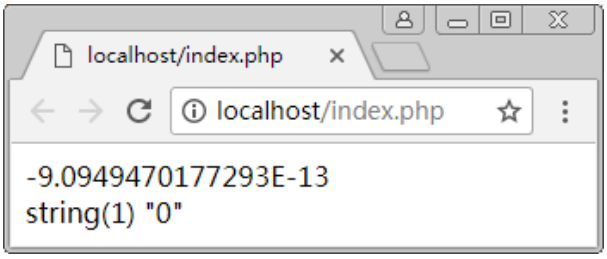
运行程序，结果如下图所示。



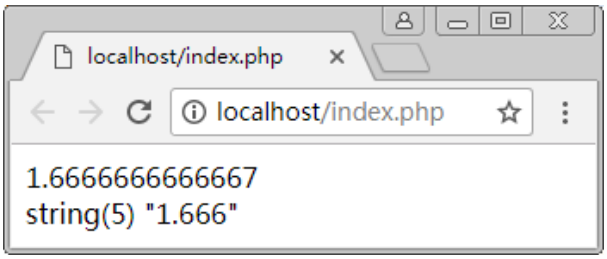
3. 利用 bcsub()进行减法运算，bcmul()进行乘法运算。

```
echo 69.1 * 100 - 6910;
echo '<br>';
var_dump(bcsub(bcmul(69.1, 100), 6910));
```

运行程序，结果如下图所示。



4. 利用 bcdiv()进行除法运算，结果如下图所示。



三. 案例总结

1. BCMath 相关函数的返回值为字符串类型。
2. 利用 BCMatch 可以解决 PHP 中的数值型表示范围有限的问题。

案例3-12 checkdate()函数

一. 案例描述

1. **考核知识点**

   日期函数

2. **练习目标**

   ➢ 掌握日期有效性的判断

3. **需求分析**

   当需要验证日期和时间的有效性时，可以使用 checkdate()函数。该函数的 3 个参数分别是整型表示的月、日、年的值。其中年的值的有效范围是 1 到 32767。

4. **设计思路（实现原理）**

   1） 利用 checkdate()函数验证一个合法日期，如 2000-12-31。

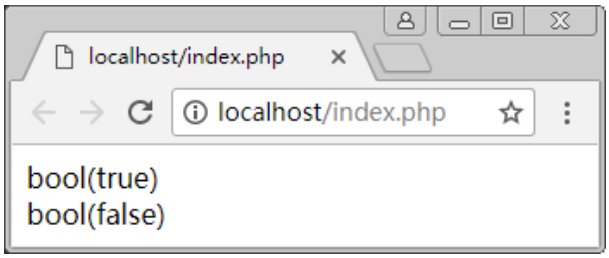
   2） 利用 checkdate()函数验证一个非法日期，如 2001-02-29。

二. 案例实现

1. 编写 index.php，具体代码如下。

```
<?php
var_dump(checkdate(12, 31, 2000));
echo '<br>';
var_dump(checkdate(2, 29, 2001));
```

2. 运行程序，结果如下图所示。



三. 案例总结

1. 利用 checkdate()函数可以很方便的检测给定日期的合法性。
2. checkdate()函数在检测日期时支持对闰年进行判断。

案例3-13 create_function()函数

一. 案例描述

1. **考核知识点**

   创建函数

2. **练习目标**

   - 掌握函数的创建

3. **需求分析**

在 PHP 中，利用 create_function()函数可以创建一个函数，改函数的第 1 个参数为被创建函数的参数字符串，第 2 个参数为函数体字符串。

4. **设计思路（实现原理）**

   1） 利用 create_function()创建一个函数。

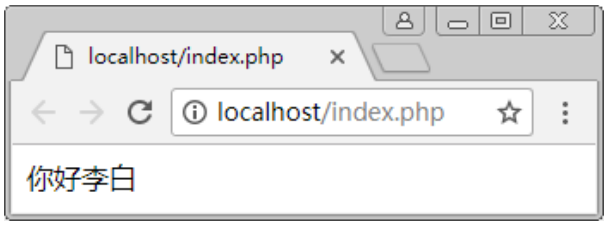
   2） 调用函数进行测试。

二. 案例实现

1. 创建 index.php，编写代码如下。

```
<?php
$lang = 'ch';
if ($lang == 'ch') {
    $fn = create_function('$name', 'echo \'你好\' . $name;');
} elseif ($lang == 'en') {
    $fn = create_function('$name', 'echo \'hello\' . $name;');
}
$fn('李白');
```

运行程序，结果如下图所示。



2. 通过上述代码创建的函数，相当于如下结果。

```
$fn = function ($name)
{
    echo '你好' . $name;
};
$fn('李白');
```

三. 案例总结

利用 create_function()函数可以在程序运行过程中创建一个函数。

案例3-14 disable_functions 配置

一. 案例描述

1. **考核知识点**

   函数的配置

2. **练习目标**

   - 掌握利用 disable_functions 禁用函数

3. **需求分析**

在 PHP 的配置文件 php.ini 中，利用 disable_functions 可以对函数进行禁用，通常用于在网站上线时，禁用一些危险的函数，以避免 PHP 脚本出现漏洞时，影响整个服务器的安全。

4. **设计思路（实现原理）**

   1） 在 php.ini 中找到 disable_functions。

   2） 禁用一些函数进行测试。

二. 案例实现

1. 在 php.ini 中，搜索“disable_functions”找到如下配置。

```
disable_functions =
```

2. 更改配置，实现禁用“phpinfo”和“exec”两个函数，用逗号“,”分隔即可。

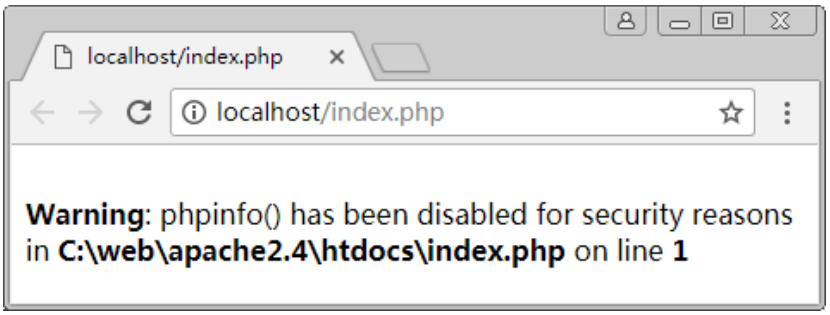
```
disable_functions = phpinfo,exec
```

更改 PHP 配置文件后重启 Apache 服务使配置生效。

3. 编写 index.php，测试被禁用的函数是否还能使用，具体代码如下。

```
<?php phpinfo();
```

运行结果如下图所示。



三. 案例总结

1. 利用 disable_functions 可以禁用一些函数。
2. 在上线环境下，建议通过 disable_functions 禁用一些危险的函数，以提升系统安全。

案例3-15 __FUNCTION__

一. 案例描述

1. **考核知识点**

   函数的使用

2. **练习目标**

   - 掌握__FUNCTION__的使用

3. **需求分析**

在 PHP 的函数中，通过“__FUNCTION__”常量可以获取当前函数的名称。

4. **设计思路（实现原理）**

   1）通过__FUNCTION__输出当前函数名称

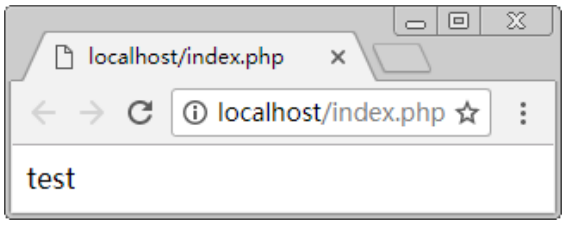
   2）通过__FUNCTION__实现递归调用

二. 案例实现

1. 编写 index.php，具体代码如下。

```
<?php
function test()
{
    echo __FUNCTION__;
}
test();
```

运行程序，结果图下图所示。



2. 通过“__FUNCTION__”还可以用来递归调用，代码如下。

```
function test($n = 2)
{
    echo $n, '-', __FUNCTION__ . '<br>';
    if (--$n > 0) {
        $fn = __FUNCTION__;
        $fn(0);
    }
}
```

三. 案例总结

利用__FUNCTION__可以获取当前函数名称。

案例3-16 进制转换

一. 案例描述

1. 考核知识点

进制转换函数

2. 练习目标

- 掌握 PHP 内置的进制转换函数

3. 需求分析

在开发中，有时需要对数值进行进制转换。通过 PHP 提供的内置函数可以轻松实现十进制、十六进制、二进制、八进制，或其他进制的转换。

4. 设计思路（实现原理）

1） 利用 decbin()、bindec()函数实现二进制与十进制的相互转换。
2） 利用利用 decoct()、octdec()函数实现八进制与十进制的相互转换。
3） 利用 dechex()、hexdec()函数实现十六进制与十进制的相互转换。
4） 利用 base_convert()函数实现任意的进制转换。

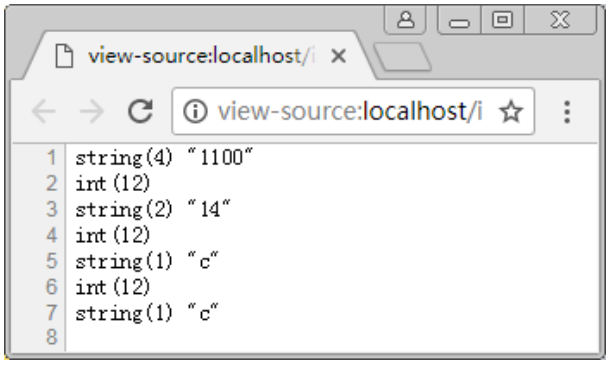
二. 案例实现

1. 编写 index.php，具体代码如下。

```
<?php
```

```
// 十进制转二进制
var_dump(decbin(12));
// 二进制转十进制
var_dump(bindec('1100'));
// 十进制转八进制
var_dump(decoct(12));
// 八进制转十进制
var_dump(octdec(14));
// 十进制转十六进制
var_dump(dechex(12));
// 十六进制转十进制
var_dump(hexdec('c'));
// 任意进制转换（将 1100 从 2 进制转换为 16 进制）
var_dump(base_convert('1100', 2, 16));
```

2. 运行程序，查看 HTML 源代码，结果如下图所示。



三. 案例总结

利用 PHP 内置函数可以轻松实现进制转换。

案例3-17 转换字符与 ASCII 码

一. 案例描述

1. **考核知识点**

   PHP 内置函数

2. **练习目标**

   - 掌握字符与 ASCII 码的转换

3. **需求分析**

利用 PHP 的内置函数可以轻松实现字符与 ASCII 码之间的转换。

4. **设计思路（实现原理）**

   1） 通过 chr()、ord()函数实现字符与 ASCII 码之间的转换。

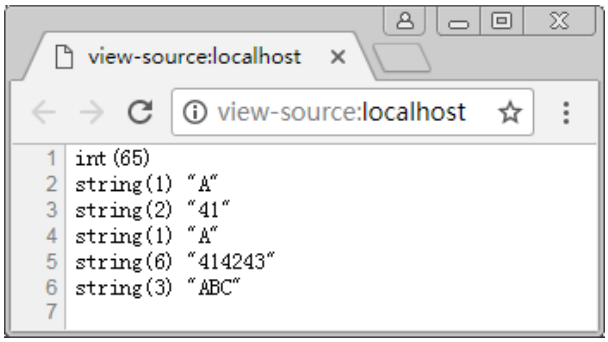
   2） 通过 bin2hex()、hex2bin()实现字符串的转换。

二. 案例实现

1. 编写 index.php，具体代码如下。

```
<?php
// 字符转 ASCII 码
var_dump(ord('A'));
// ASCII 码转字符
var_dump(chr(65));
// 字符转十六进制 ASCII 码
var_dump(bin2hex('A'));
// 十六进制 ASCII 码转字符（PHP >= 5.4）
var_dump(hex2bin('41'));
// 将字符串数据转十六进制
var_dump(bin2hex('ABC'));
// 将十六进制数据转字符串
var_dump(hex2bin('414243'));
```

2. 运行程序，查看 HTML 源代码，结果如下图所示。



三. 案例总结

转换一个字符时，使用 chr()、ord()函数；转换整个字符串时，使用 bin2hex()、hex2bin()函数。

案例3-18 ASCII 字符转 HTML 实体字符

一. 案例描述

1. **考核知识点**

   内置函数

2. **练习目标**

   - 掌握 ASCII 字符与 HTML 实体字符的转换

3. **需求分析**

在网页中通过 PHP 输出 HTML 标记（如“ ”）时，会被浏览器解析，这会导致尖括号等字符无法原样显示。为此，可以将一些字符转换成 HTML 的实体字符表示方法。

4. 设计思路（实现原理）

1）获取字符串中的每一个字符。

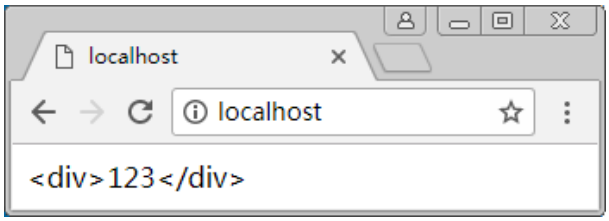
2）将字符转换成 HTML 实体字符。

二. 案例实现

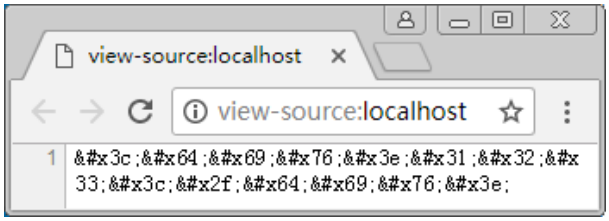
1. 编写 index.php，具体代码如下。

```
<?php
function hexentities($str) {
    $result = '';
    for ($i = 0, $len = strlen($str); $i < $len; ++$i) {
        $result .= '&#x' . bin2hex($str[$i]) . ';';
    }
    return $result;
}
echo hexentities('<div>123</div>');
```

2. 运行程序，结果如下图所示。



3. 查看 HTML 源代码，结果如下图所示。



三. 案例总结

利用 bin2hex()可以将 ASCII 字符转换成十六进制 ASCII 码，然后拼接“&#x”即可转换成 HTML 实体字符。